VHDL Fault Simulation and Automatic Test Pattern Generation Requirements Document

Contract No. F30602-95-C-0220

Submitted to:

Rome Laboratory/ERM
525 Brooks Rd.
Griffiss AFB, New York 13441-4505

Attention: Dr. James P. Hanna

Submitted by:

Barry W. Johnson
Professor of Electrical Engineering

D. Todd Smith
Research Associate

Todd A. DeLong
Research Scientist

SEAS Report No. UVA/525824/EE96/101
January 1996

DEPARTMENT OF ELECTRICAL ENGINEERING

SCHOOL OF
ENGINEERING
& APPLIED SCIENCE

University of Virginia
Thornton Hall
Charlottesville, VA 22903

UNIVERSITY OF VIRGINIA
School of Engineering and Applied Science

The University of Virginia's School of Engineering and Applied Science has an undergraduate enrollment of approximately 1,500 students with a graduate enrollment of approximately 600. There are 160 faculty members, a majority of whom conduct research in addition to teaching.

Research is a vital part of the educational program and interests parallel academic specialties. These range from the classical engineering disciplines of Chemical, Civil, Electrical, and Mechanical and Aerospace to newer, more specialized fields of Applied Mechanics, Biomedical Engineering, Systems Engineering, Materials Science, Nuclear Engineering and Engineering Physics, Applied Mathematics and Computer Science. Within these disciplines there are well equipped laboratories for conducting highly specialized research. All departments offer the doctorate; Biomedical and Materials Science grant only graduate degrees. In addition, courses in the humanities are offered within the School.

The University of Virginia (which includes approximately 2,000 faculty and a total of full-time student enrollment of about 17,000), also offers professional degrees under the schools of Architecture, Law, Medicine, Nursing, Commerce, Business Administration, and Education. In addition, the College of Arts and Sciences houses departments of Mathematics, Physics, Chemistry and others relevant to the engineering research program. The School of Engineering and Applied Science is an integral part of this University community which provides opportunities for interdisciplinary work in pursuit of the basic goals of education, research, and public service.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	January 1996	Progress 9/95 - 12/95

4. TITLE AND SUBTITLE
VHDL Fault Simulation and Automatic Test Pattern Generation Requirements Document

5. FUNDING NUMBERS
Grant No. F30602-95-C-0220

6. AUTHORS(S)
Barry W. Johnson, D. Todd Smith, Todd A. DeLong

7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(ES)
University of Virginia
Department of Electrical Engineering
School of Engineering and Applied Science
Thornton Hall
Charlottesville, VA 22903-2442

8. PERFORMING ORGANIZATION REPORT NUMBER
UVA/525824/EE96/101

9. SPONSORING/MONITORING AGENCY NAMES(S) AND ADDRESS(ES)
Rome Laboratory/ERM
525 Brooks Rd.
Griffiss AFB NY 13441-4505

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The view, opinion, and/or findings contained in the report are those of the authors and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other documentation.

12a. DISTRIBUTION/AVAILABILITY STATEMENT

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

See report.

14. SUBJECT TERMS

15. NUMBER OF PAGES
29

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	Unlimited

1. Introduction

This document describes the requirements for a design automation tool that performs fault simulation and fault grading using the Very High Speed Integrated Circuit (VHSIC) Hardware Description Language (VHDL) [1]. The goal is to develop a tool which can automatically perform fault grading using fault simulation on any VHDL model using any fully compliant VHDL simulator. The current state of the art for VHDL-based fault simulation is very limited. Preliminary research has been performed for fault insertion techniques, but tools do not exist which perform VHDL fault simulation/fault grading [2]. Reference [2], developed to assist in the preparation of this requirements document summarizes the following items: (1) fault simulation techniques described in the literature, (2) VHDL-based fault simulation techniques, (3) commercial fault simulation products which accept VHDL models as input and perform fault simulation without a VHDL simulator, (4) fault grading techniques, and (5) test pattern generation methods. The term VHDL-based fault simulation is defined in this document as a fault simulation technique which uses a fully compliant VHDL simulator to perform fault simulation. There exist numerous commercial products which accept VHDL models as input and perform fault simulation. However, these products perform fault simulation using a proprietary (nonVHDL) simulation engine. This type of commercial fault simulation tool is a nonVHDL-based fault simulation technique. The extensive review of the state of the art has revealed that VHDL-based fault simulation tools do not exist.

The fault grading process entails evaluating all relevant faults for the Device Under Test (DUT) represented by the VHDL model for a set of input patterns. The percentage of detected faults is referred to as fault coverage. The set of input patterns used to exercise the DUT to reveal faults is typically generated by some Test Pattern Generation (TPG) technique. TPG methods fall into three broad categories: (1) Automatic TPG (ATPG), (2) Random TPG (RTPG), and (3) Manual TPG (MTPG). The requirements of the fault grading toolset are derived to augment the TPG process. Typically, TPG techniques utilize fault simulation to determine what faults are detected by a given input vector. For example, ATPG generates an input vector to detect a class of equivalent faults. The derived input vector more than likely will detect other faults. Fault simulation is employed to determine what additional faults are detected by the derived input vector.

This requirements document describes the desired attributes of a fault simulation/fault grading tool. The purpose of defining the desired requirements for the fault simulation/grading toolset is to clearly establish a framework for the ideal tool set. The scope of the requirements may be such that only a subset of the desired features/attributes can be developed with the current research effort. The requirements document is the first stage of a three stage tool design process. The requirements document defines the attributes/features which are desired for a VHDL-based fault simulation/fault

grading tool. The second development stage consists of writing a specification document which details the implementation plan which satisfies the requirements described in this document. The specification document is referred to as an implementation plan which is scheduled to be completed 1 April 1996. The goal of the implementation plan is to describe the design of the fault simulation/fault grading tool in sufficient detail such that a team of independent software developers could construct the tool. Additionally, the implementation plan will define the attributes/features of the VHDL-based fault simulation/fault grading tool which are scheduled to be implemented. The third stage of the development effort consists of building the tool set defined by the specification and writing a users manual and installation manual. The requirements document is also used at the end of the tool development process to verify that the developed tool has the desired characteristics.

Some relevant background information concerning the design process is introduced before the requirements for the fault simulation/fault grading tool are presented. The design cycle for electronic devices is becoming shorter while the demand for quality is increasing. The decreasing time to market forces numerous improvements in the design process. One improvement is the use of an integrated set of design tools during all stages of the design process. Ideally, a single modeling language is used to represent a design throughout all levels of design refinement. A standard Hardware Description Language (HDL) such as VHDL provides a single modeling language for all levels of design abstraction.

A standard HDL provides several attributes which promote design process efficiency. Five of the key attributes which speed the design process are: (1) the design can be modeled with a single language during all levels of abstraction, (2) components of a design can be represented at different levels of abstraction while maintaining a consistent system level model, (3) iterative design refinement is supported for each component, (4) design hierarchy is encapsulated in a consistent fashion, and (5) virtual prototyping of the system is possible at each step of the design process.

Several diagrams are included to assist the reader in understanding the aforementioned five attributes. Figure 1.1 depicts the concept of virtual prototyping of a design throughout the design process. The design process begins with a system definition which is indicated by the leftmost block in Figure 1.1. The design of the device begins at a high-level of abstraction; that is, the function is designed first. Once the functional design is complete then the design is partitioned into hardware and software components. The hardware and software partitions are further refined and verified for correctness using virtual prototyping. The hierarchical nature of the design process along with the iterative refinement concepts are illustrated in Figure 1.2. Three levels of design abstraction are illustrated by the architectural, algorithmic, and logic layers in Figure 1.2. The design process is described in three stages; that is, definition, hardware and software codesign, and implementation. The three phases of the design process are shown by three regions in Figure 1.2.

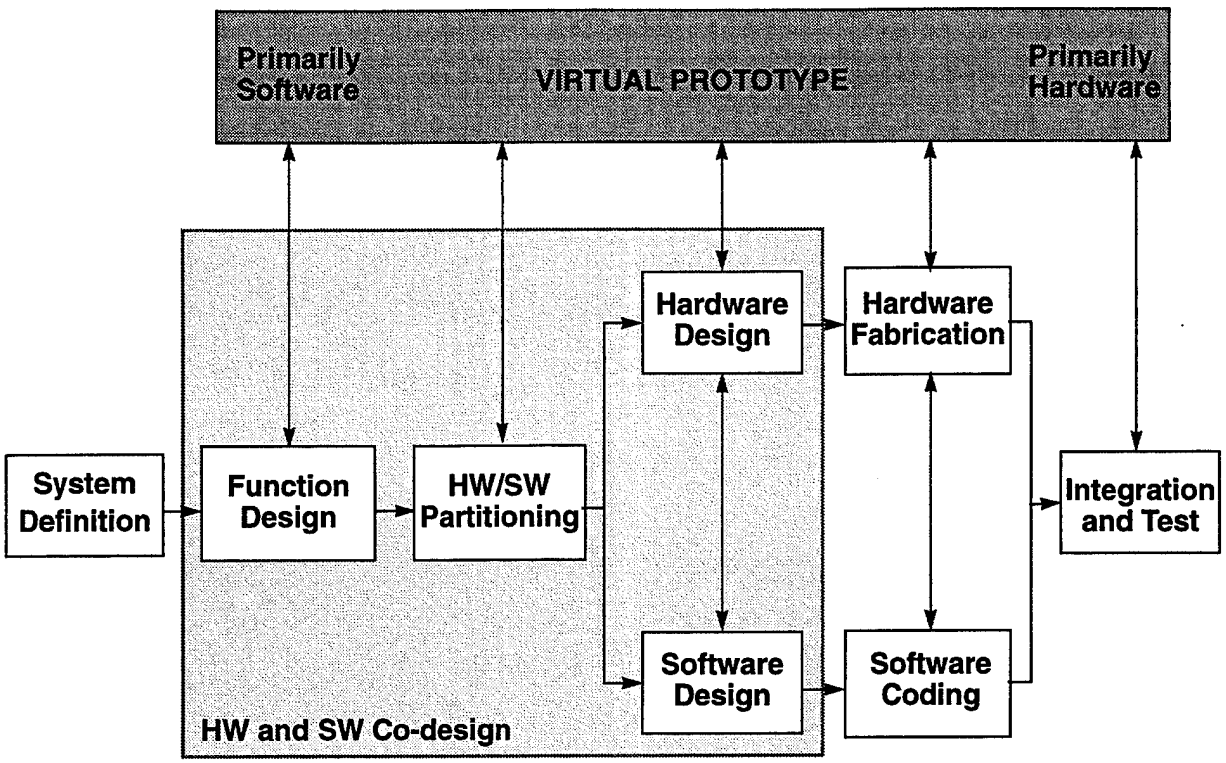


Figure 1.1. Block diagram depicting a design process which utilizes virtual prototyping (from [6]).

As the design is partitioned and refined the design knowledge is increased. The information content associated with the design increases as the design progresses from the definition phase to the implementation phase. Likewise, each partition of the design can be refined and further subdivided as the design is completed. The iterative refinement of a design partition is indicated by the columns in Figure 1.2.

The design process should also support hybrid modeling. A hybrid model is a model which contains one or more components whose functional map is unknown, but the timing characteristics are specified. An example is provided to illustrate the concept of hybrid modeling. Consider the case where a Digital Signal Processing (DSP) application is being developed. The algorithm which defines the filtering function which is performed by the DSP processor is known. However, the functional mappings associated with the components which read inputs and deliver outputs to and from the system are unknown. The maximum allowable delay associated with the system inputs and outputs is known. A graphical depiction of this example hybrid model is included as Figure 1.3. The input and output components of the system are depicted by the left and right most boxes in Figure 1.3. The input/output blocks contain a **U** to indicate that the functional mapping is undefined. The DSP algorithm is represented by the **D** block in Figure 1.3 where the **D** denotes the functional map of the component is defined. One of the attributes common to a hybrid model is that a translation is required when entering the defined function domain from the undefined function

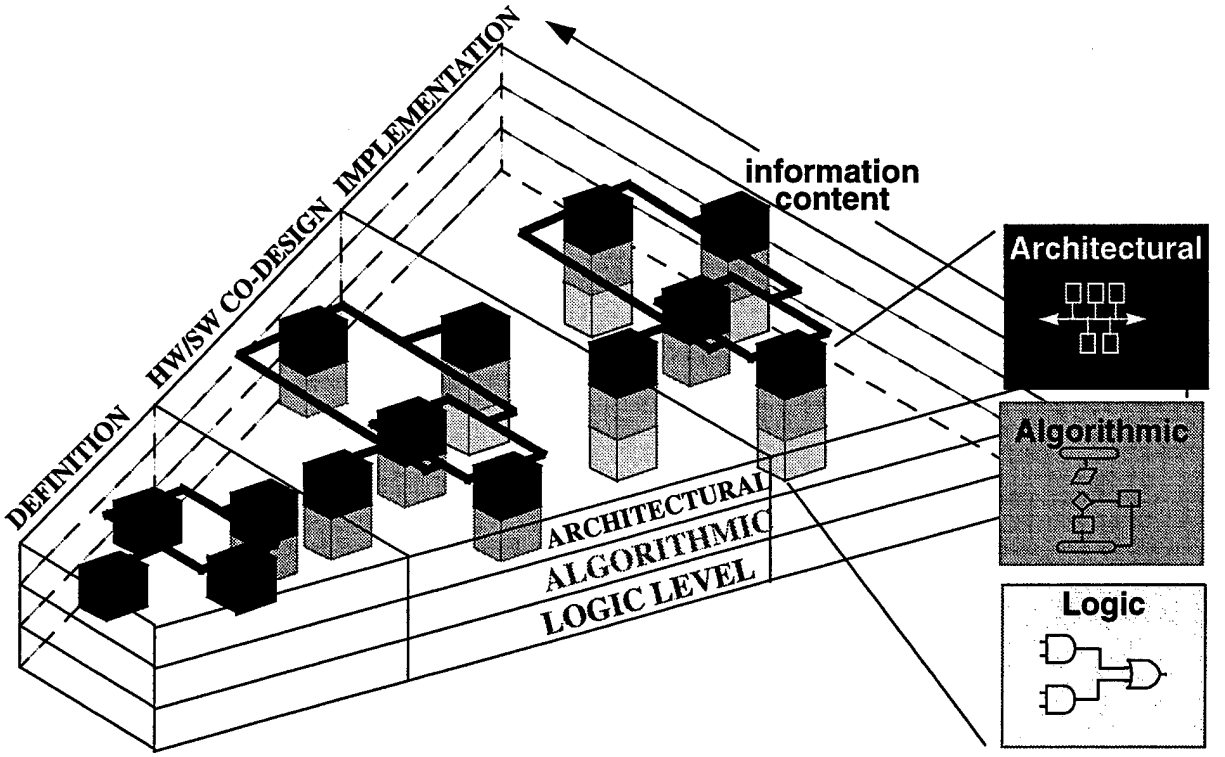


Figure 1.2. Diagram depicting the relationship between the design process and the information content of the design (from [6]).

domain. This translation is represented as a **U/D** block in Figure 1.3. The input information provided by the **U/D** block is then processed by the **D** block using the model of the DSP filter to determine the appropriate output. A translation is necessary to convert the information produced by the DSP filter to the **U** domain. The transition from the **D** to **U** domain is performed by the Evaluator block in Figure 1.3.

A hybrid model allows for performance data to be gathered on a design which contains both **U** and **D** domain components. Additionally, the performance and function of **D** domain components is tested during the simulation of a hybrid model. As the design is further refined eventually all components in the design become **D** domain components. At each level of model refinement the hybrid model provides the designer useful information concerning both the performance and function of the design. This information assists the designer in making further design refinements.

Now that the attributes of the ideal design process are described the requirements associated with the ideal fault simulation/fault grading tool are introduced. The goal is for the fault simulation/ fault grading tool to be seamlessly integrated into the aforementioned design process. Specifically, fault simulation can be performed at each level of design refinement throughout the design process. The design refinement process is depicted in Figure 1.1 while the level of model refinement throughout the design process is shown in Figure 1.2. Ideally, fault simulation can be performed at each stage of the design process depicted in Figure 1.1. Likewise, the fault simulation tool should

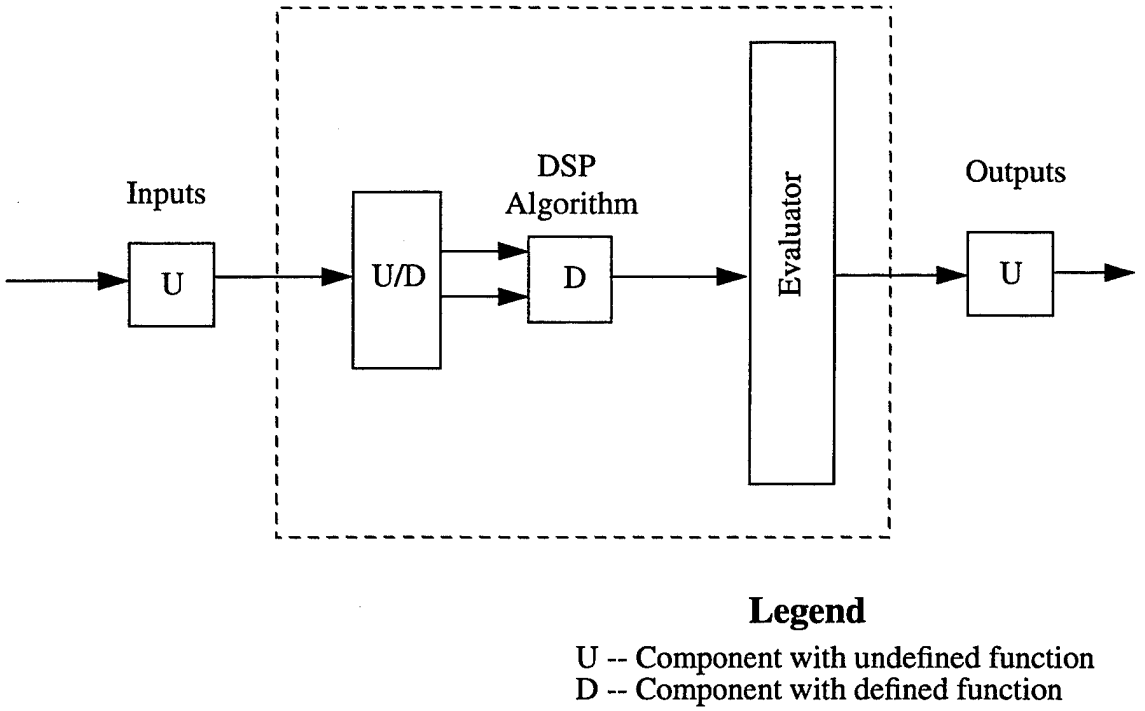


Figure 1.3. Hybrid model example showing all major components.

accept models which contain mixed-levels of abstraction. Figure 1.2 illustrates the various types of mixed level models which are possible throughout the design process.

This requirements document is organized in the following fashion. Following this brief introduction, Section 2 provides a description of the set of requirements for the VHDL-based fault simulation tool. A list of requirements for a fault grading tool is described in Section 3. A description of tool structure requirements is included as Section 4. Concluding remarks along with a summary of all requirements for the VHDL-based fault simulation/fault grading tool is included as Section 5.

2. Fault Simulation Requirements

The individual fault simulation requirement attributes are described in each of the following subsections. Each requirement subsection has a standard format which consists of two major components: (1) definition of requirement and (2) justification of the requirement.

2.1. Unified Modeling Methodology using VHDL

Requirement Definition

The DUT must be modeled using a hardware description language that allows all levels of design to be modeled. VHDL is the required modeling language. Specifically, the IEEE standard (1076-1993) for VHDL must be utilized. Throughout the remainder of this document whenever the term VHDL is used it is implied that the VHDL must be compliant with the 1076-1993 standard.

Additionally, the signal values in the VHDL model must adhere to the IEEE standard (1164-1993) for Multivalue Logic System for VHDL Model Interoperability [3]. This standard is commonly referred to as the MultiValue Logic 9 (MVL9) because the IEEE 1164-1993 standard defines nine standard logic values.

Requirement Justification

Electronic systems designed for the Department of Defense (DOD) must be modeled in VHDL. Designs described by VHDL allow the DOD to use any VHDL-based fabrication house to manufacture additional systems. The ability to build additional system components independent of the original manufacturer is very important to the DOD. The life cycle of DOD systems often times exceeds 25 years. The DOD needs to be able to obtain additional systems from other sources without redesign in the event that the original system manufacturer goes out of business.

A VHDL-based design philosophy allows each component to be developed independently of every other component. The interface between each component must be well defined for the independent component design to be possible. For instance, some component models could be purchased off the shelf from a digital design house. The use of the MVL9 standard also promotes independent design efforts because all signal types that can be used in the interface of two components are clearly defined. Another benefit to a VHDL-based approach is the ability to generate component libraries of often used components.

2.2. Fault Insertion Technique

Requirement Definition

A fault simulation engine performs fault simulation by inserting faults into the model of the DUT and then simulating the faulty DUT. The fault insertion technique for VHDL-based fault simulation has several required attributes: (1) the fault insertion method must be performed in a consistent fashion at all levels of modeling abstraction, (2) the fault insertion mechanism must be decoupled from the model as much as possible to minimize changes to the model, (3) fault insertion must be performed using standard VHDL features, and (4) the fault insertion technique must be simulator independent.

Requirement Justification

A fault insertion mechanism which satisfies the aforementioned four attributes provides several benefits. Having a consistent fault insertion mechanism allows for fault simulation to be performed on a VHDL model at any level of abstraction. In fact, fault simulation of a mixed-level VHDL model is possible. Decoupling the fault insertion technique from the structure of the model allows for the fault insertion technique to be readily added to existing VHDL models in a straight forward fashion. Ideally, the underlying function and structure of a VHDL model need not be known to add the decoupled fault insertion method. Using standard VHDL attributes to perform the fault inser-

tion allows for the fault simulation to be performed on any VHDL compliant simulator. The ability to use any VHDL simulator allows the designer to migrate to different design environments while maintaining a VHDL-based fault simulation capability. Additionally, performing the fault insertion using standard VHDL features allows the same simulation environment that was used to verify the function and performance of the DUT to be used for fault simulation. The use of the same simulation engine for all phases of the design streamlines the design process by minimizing the number of different disjoint tools that the design engineer must master to complete the design.

2.3. Automated Fault Simulation

Requirement Definition

The fault grading process of any nontrivial DUT typically requires the evaluation of a significant number of faults. The evaluation of a large number of faults may translate into a significant number of fault simulations. For fault grading to be performed in an efficient fashion it is required that a large number of fault simulations be performed in an automated fashion.

Requirement Justification

The evaluation of a large set of faults in an automated batch fashion is essential for making the fault grading process feasible. The design engineer should not be required to interactively run a fault simulation for each individual fault of interest due to the amount of time required to set up, run, and gather the results for each interactive fault simulation.

2.4. Interactive Fault Simulation

Requirement Definition

The fault simulation engine must be able to evaluate a single fault in an interactive mode.

Requirement Justification

For faults which are not detected, it is a desirable feature to allow the designer to interactively observe the effect of the fault on the DUT. The interaction of the fault with the DUT can provide insight to the designer for: (1) the selection of additional input test vectors, and (2) performing model refinement so that the undetected fault condition is removed.

2.5. Fault Model

Requirement Definition

The fault simulation tool must support designer specified fault models for each level of modeling abstraction. The fault simulation tool must support the stuck-at fault model for gate-level models.

Requirement Justification

At different levels of design abstraction both the function and the faulty behavior of the component are represented differently. For example, consider a component which is represented by a

behavioral model. The fault model for the behavioral model must be consistent with the level of abstraction associated with the component. The use of a gate-level fault model to describe the faulty behavior of a behavioral model is one example of a fault model having a different level of abstraction than the component model and therefore causing a modeling inconsistency. The fault simulation tool must allow the designer to specify the fault model which is appropriate for a given model at a given level of abstraction.

2.6. VHDL Fault Simulation using Design Hierarchy

Requirement Definition

The fault simulation process can be accelerated by exploiting the hierarchy present in the design of the DUT. The fault simulation tool is required to utilize the design hierarchy where possible using hierarchical fault simulation techniques to speed the fault simulation process. Specifically, hierarchical fault simulation entails having all but one component in a VHDL model modeled with a high-level behavioral model. The component which is being evaluated using fault simulation is modeled at the lowest level of abstraction such as, a structural model. All components in the DUT must have a defined functional map or stated another way must exist in the **D** domain for hierarchical fault simulation to be possible. Once all the faults associated with the low-level component model are evaluated then another component is selected for evaluation. The high-level model of the selected component is removed from the DUT model and replaced with the low-level structural model. The previously evaluated low-level structural model is removed from the DUT and replaced with a high-level behavioral model. The fault simulation process is then continued.

Requirement Justification

The fault simulation of a nontrivial DUT requires significant amounts of computational resources. The fault grading of a nontrivial DUT will require a significant number of fault simulations. For this reason it is imperative to make the fault simulation process as computationally efficient as possible. Exploiting design hierarchy during fault simulation is one technique for increasing the fault simulation efficiency. It is envisioned that the hierarchy of the device will be exploited for the automated evaluation of a large number of faults. The hierarchical fault simulation technique described in [2] exploits hierarchy in an efficient fashion. Hierarchical fault simulation creates a unique DUT model for each design partition. The basic concept is to have all partitions in the design represented at the highest level of abstraction except for the component which is being evaluated using fault insertion. The evaluated component is represented at the lowest possible level of abstraction such as the gate level. The use of design hierarchy in this fashion reduces the computational cost of fault simulation because all design partitions except one are evaluated using a high-level model. The computational cost associated with evaluating a high-level

model such as a behavioral model is always less than a low-level model such as a gate-level model with timing. The automated fault simulation requirement is described in Section 2.3.

2.7. Input to Output Representation of the Test Vector Set

Requirement Definition

The input and the corresponding output for a given DUT must be known in advance before fault grading can be performed. An IEEE standard (1029.1-1991) for Waveform and Vector Exchange (WAVES) [4] must be used to store the input vector stimulus to the DUT and the correct output response for the DUT. It is envisioned that existing Rome Laboratory tools can be used to generate the WAVES data set.

Requirement Justification

WAVES allows for the input vectors and the output response to be specified in VHDL so that the functional input to output mapping of the DUT is specified. The WAVES interface automatically detects when simulated output does not match a stored output. The notification of a mismatch is used by the fault simulation tool to determine that a fault is detected.

The input patterns to test the DUT are generated using some Test Pattern Generation (TPG) strategy. There are three commonly used approaches: (1) Automated Test Pattern Generation (ATPG), (2) Random Test Pattern Generation (RTPG), and (3) Manual Test Pattern Generation (MTPG) [2]. The fault simulation/fault grading tool should augment the TPG techniques in a seamless fashion. The TPG tool requires a very specific set of information from the fault simulation tool; that is, the list of detected and undetected faults associated with a given input vector. Likewise, the information required by the fault simulation tool is: (1) the DUT model, (2) the input test vector, (3) the corresponding output of the DUT, and (4) the set of faults. The seamless integration of TPG and fault simulation occurs when the file input/output formats common to both tools are clearly specified. Specifically, the use of WAVES standardizes the exchange of the input vector and corresponding output between the TPG tool and the fault simulator. The DUT model format is specified to be VHDL for both the TPG tool and the fault simulator. The only remaining file format specifications are: (1) the fault list to fault simulate which is generated by the TPG tool and read by the fault simulator, and (2) the list of detected and undetected faults which is generated by the fault simulator and read by the TPG tool.

2.8. Fault Simulation Acceleration

Requirement Definition

The fault simulation tool must incorporate the use of fault equivalency to accelerate the fault grading process where applicable.

Requirement Justification

Exploiting fault equivalency in the DUT can significantly reduce the number of fault simulations. Several fault equivalency techniques exist [2]. For example, Critical Path Tracing (CPT) is one gate-level fault equivalency method which may be incorporated into the fault simulation tool. Fault equivalency techniques can enable the designer to evaluate all faults associated with the DUT in a timely fashion.

2.9. Fault Simulation Output Data

Requirement Definition

The VHDL-based fault simulation technique is required to provide all relevant information necessary to perform fault grading. The required output set consists of: (1) the set of all faults detected during fault simulation, (2) the set of all faults undetected by the fault simulation, and (3) the number of detected/undetected faults for the fault simulation.

Requirement Justification

The fault simulation tool must provide the data that is necessary to perform fault grading.

2.10. Synthesis

Requirement Definition

The fault simulation tool must be able to evaluate a VHDL model before and after synthesis. Currently, synthesis is performed at the behavioral level of the design process; that is, a behavioral VHDL model is used to synthesize a gate-level VHDL model. The proposed IEEE standard (P1076.4) VHDL Initiative Towards Application Specific Integrated Circuit (ASIC) Libraries (VITAL) will provide a mechanism to back-annotate the timing information of a synthesized ASIC into a VHDL model. The fault simulation tool must be able to fault simulate: (1) the presynthesis model, (2) the synthesized VITAL/nonVITAL compliant model without timing, and (3) VITAL/nonVITAL compliant model with timing information.

Requirement Justification

Synthesis is one accepted technique that can be employed to decrease the time required to design a device. For synthesis to produce a testable device it is essential that the designer be able to fault grade the VHDL model before and after synthesis. A graphical depiction of the VITAL synthesis process is included as Figure 2.1. A synthesizable behavioral model is represented by the top most block in Figure 2.1. The synthesis tool utilizes a standard VITAL library and a technology dependent library during the generation of the structural VHDL model. The output of the synthesis tool is a structural VHDL model without timing depicted as the lower left rectangular block in Figure 2.1. A placement and routing tool is used to generate timing information. The timing data is then back annotated into the VITAL model using a standard delay format. After the back anno-

tation is complete then a VITAL compliant model with timing is generated as indicated by the lower rightmost rectangular block in Figure 2.1.

Three different VHDL models are available during the VITAL compliant synthesis process: (1) the behavioral VHDL model, (2) the structural VHDL model without timing, and (3) the structural VHDL model with timing. Fault simulation/fault grading can be performed on each VHDL model. Fault simulation/fault grading of the structural model with timing is required to accurately estimate the fault and/or error latency associated with a component. Typically, fault/error latency information is required for the evaluation of the reliability and/or safety of an ultra dependable system. Thus, the fault simulation of the structural model with timing information is required to accurately measure the dependability of the DUT.

2.11. Tool Host Environment

Requirement Definition

The fault grading/fault simulation tool must execute on Sun-based workstations to include the following model types: (1) SPARC® 2, (2) SPARC® 10, and (3) SPARC® 20[1].

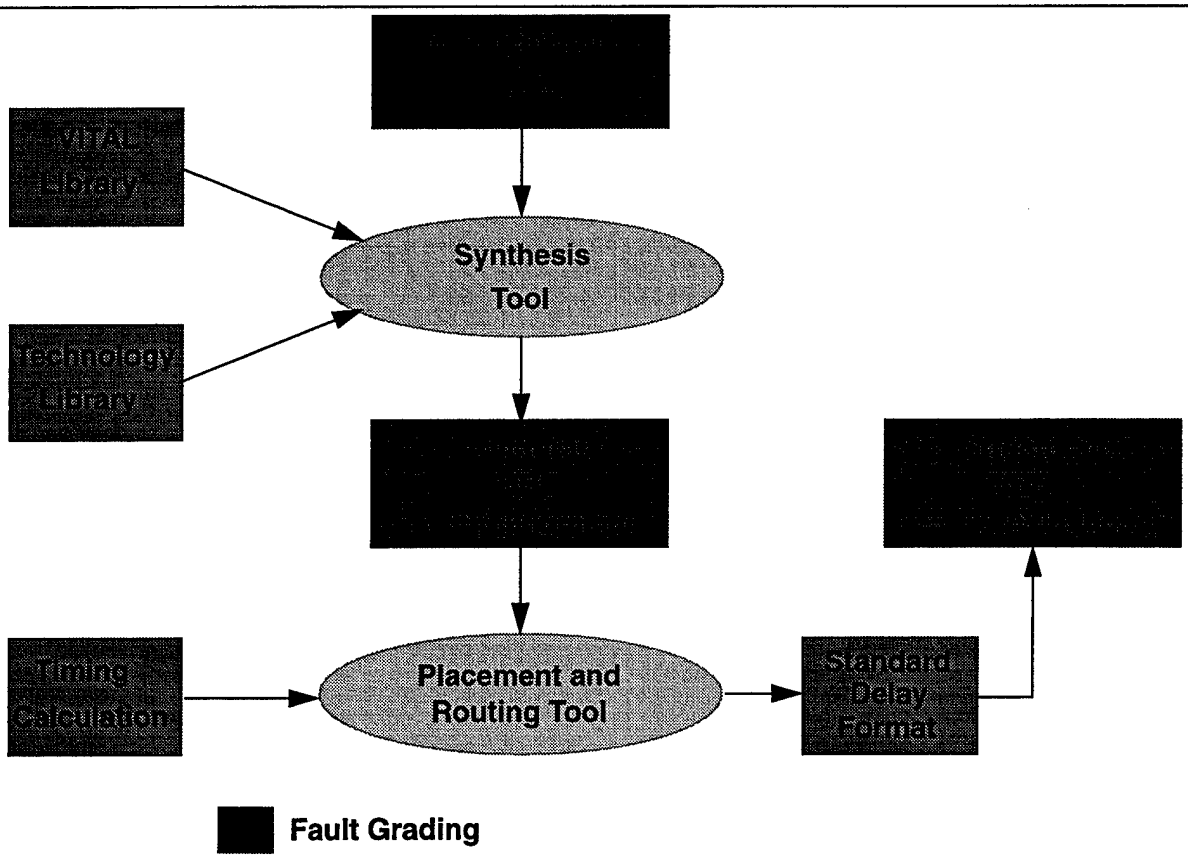


Figure 2.1. Block diagram depicting the relationship between the synthesis process and fault grading.

1. SPARC is a registered trademark of SPARC International, Inc.

Requirement Justification

The host workstation requirement is derived from the fact that Rome Laboratory owns a large base of existing design tools that are hosted on SPARC® workstations.

2.12. Hybrid Modeling Fault Simulation

Requirement Definition

The fault simulation tool must support the representation of the DUT by hybrid models. Specifically, the fault simulation tool must support the injection of faults into components which reside either in the **D** or **U** domain.

Requirement Justification

The designer must be able to evaluate the behavior of the DUT during each stage of the design process. The insertion of faults into a hybrid model allows the designer to evaluate the design at each level of design refinement.

3. Fault Grading Requirements

One of the goals of a fault simulation tool is to provide information which is used to perform fault grading. The proposed tool set is required to perform fault grading in an automated fashion. The end value produced by the fault grading process is a fault coverage estimate. Fault coverage is the conditional probability of detecting a fault for a predefined set of test vectors given that a fault exists in the DUT. The coverage estimate is obtained by dividing the number of detected faults by the total number of evaluated faults.

The individual requirements for the fault grading process are described in the following subsections. The subsections are organized using a predefined format. Specifically, the requirement is stated first and then the rationale justifying why the requirement is needed is presented next.

3.1. Automated Fault Grading

Requirement Definition

The fault simulation and fault grading process must be performed in a totally automated fashion. No interaction is required by the designer with the tool once the set of faults and the input vector set are defined.

Requirement Justification

The fault grading process can require a large amount of time to perform. Thus, it is an inefficient use of the designer's time to have the designer perform fault grading in an interactive fashion.

3.2. Information Required for Fault Grading

Requirement Definition

The fault grading process requires that the fault simulation tool provide information concerning the detection status of faults associated with the DUT. Specifically, the fault simulation tool must: (1) list the set of detected and the set of undetected faults associated with a set of fault simulation experiments, and (2) provide the number of detected/undetected faults for a set of fault simulation experiments. This information is used to perform coverage estimation.

Requirement Justification

The data required for fault grading must be clearly defined to assure that the fault simulation tool provides the necessary fault grading information.

3.3. Fault Grading Methodology

Requirement Definition

The fault grading methodology employed by the fault grading tool must adhere to the process outlined in MIL-STD 883 [5]. The fault grading process for gate-level models is completely described by MIL-STD 883. Specifically, MIL-STD 883 describes a method for constructing a minimum fault set for a gate-level model based on the single stuck-at fault model and the use of fault equivalency. Fault set construction rules are provided to assure that one fault from an equivalent fault set is included in the list of faults to evaluate. The coverage estimate is obtained by utilizing one of two techniques. The first coverage estimation method requires that all faults in the fault list of interest must be evaluated. The second coverage estimation method entails randomly sampling the fault list of interest and estimating coverage using a statistical approach. The statistical coverage estimate is evaluated at a 95% confidence level to determine the confidence interval of the coverage point estimate.

MIL-STD-883 also provides a framework for evaluating models which are represented at a higher level of abstraction than the gate level. For models at higher levels of abstraction the designer is responsible for: (1) defining and justifying the fault model, and (2) defining and justifying the test strategy. Coverage estimation for high-level models is performed in an identical fashion as coverage estimation of gate-level models. Specifically, the coverage estimate is obtained by dividing the number of detected faults by the number of evaluated faults. Either the complete fault set can be evaluated or a random sample of the fault set is evaluated. If the sampling approach is taken then a confidence interval is calculated for a 95% confidence level.

Requirement Justification

All electronic devices which are manufactured for the United States (US) Department of Defense (DOD) and are required to be tested at the time of manufacture must utilize the fault grading methodology specified in MIL-STD 883.

4. Tool Structure Requirements

The requirements associated with the structure of the fault simulation/fault grading tool are presented in this section. The basic structure of a VHDL-based fault simulation tool is included as Figure 4.1 to facilitate discussion. There are several items of information that the designer must provide the fault simulation tool: (1) a model of the DUT, (2) the input test vectors and the correct output response of the DUT for each input, and (3) the fault model. For the proposed tool the input/output data is stored using the WAVES standard. The WAVES specific information is represented by dashed blocks in Figure 4.1. The fault simulation tool consists of five major components: (1) a VHDL simulator, (2) a complete fault list generator, (3) a preprocessing fault equivalence program which generates the working fault list, (4) a post processing fault equivalence program, and (5) a fault injection process. The working fault list contains the set of unevaluated faults. Also, Figure 4.1 represents the structure required for the exhaustive fault grading approach. If statistical fault grading is desired then a random sampling process block needs to be inserted between the preprocessing fault equivalence block and the working fault list block in Figure 4.1. The designer is also allowed to fully specify the fault list of interest. The fault model block and the generate fault list block in Figure 4.1 are replaced with a user defined fault list block for the case where the designer defines the fault list of interest.

The following subsections describe the tool structure requirements. Specifically, Section 4.1 describes the commercial Computer Aided Design (CAD) tool independence requirements. Fault simulation/fault grading DUT input file format requirements are presented in Section 4.2. The tool process structure requirements are described in Section 4.3. Likewise an overview of existing fault simulation techniques developed in [2] is included as Section 4.4.

4.1. Commercial Computer Aided Design Tool Independence

Requirement Definition

The fault simulation fault grading tool must be designed to be independent to the greatest extent possible of the commercial CAD tool which is used to design the DUT. The fault simulation/fault grading tool must be integrated into a commercial CAD tool to allow for a seamless design environment. Designing the fault simulation tool to be independent from a commercial CAD tool minimizes the effort required to integrate the fault simulation/fault grading tool into a commercial CAD tool.

Requirement Justification

CAD tool independence allows the fault simulation/fault grading tool to be easily integrated into any commercial CAD tool. The ease of integration is essential to allow the fault simulation/fault grading tool to be readily added to commercial CAD tools.

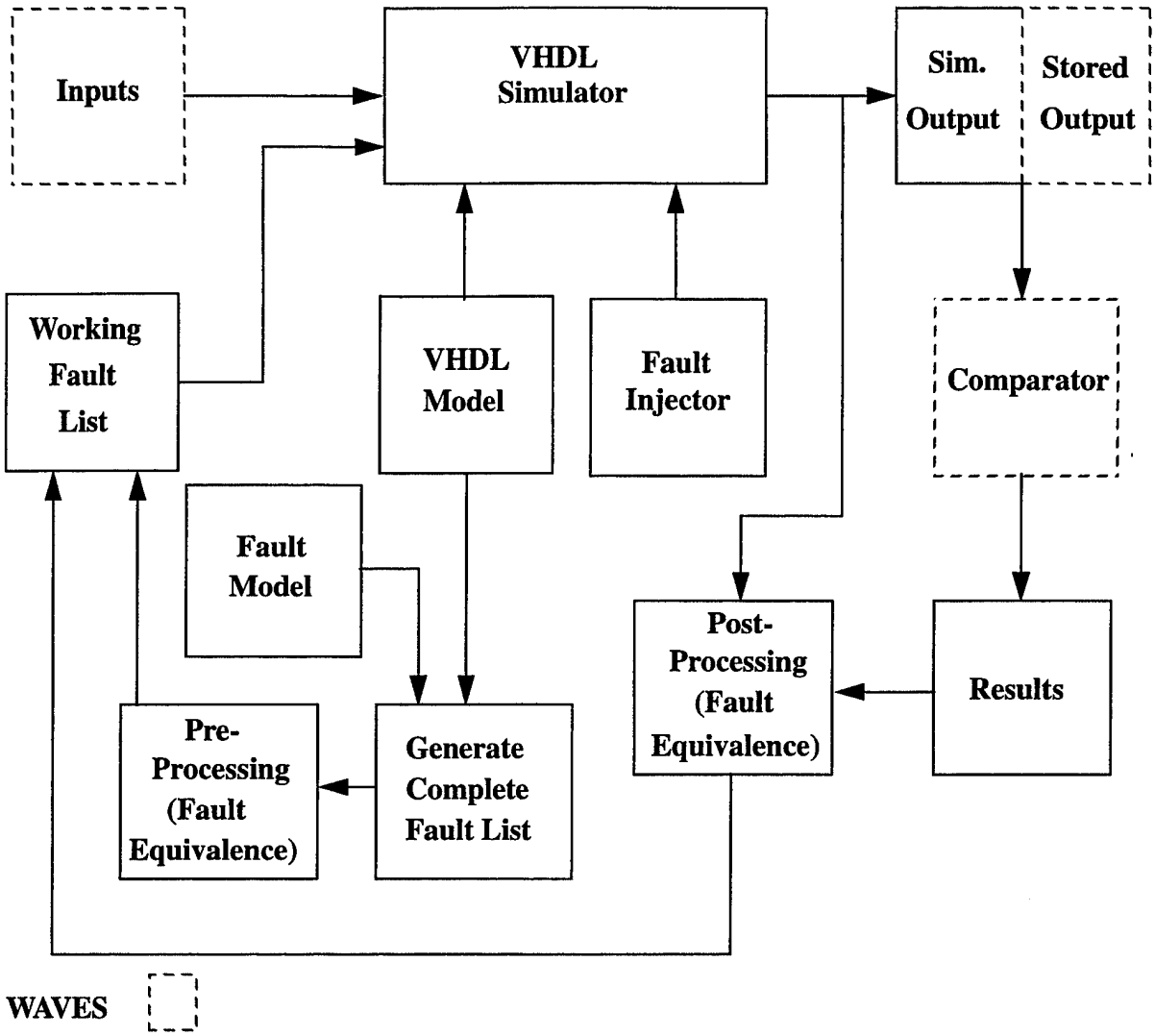


Figure 4.1. Block diagram of all key structural components of a VHDL-based fault simulator.

4.2. DUT Input File Format

Requirement Definition

The DUT input file format for the fault simulation/fault grading tool is either a VHDL file or an Electronic Data Interchange Format (EDIF) revision 3.0.0 file.

Requirement Justification

Most commercial CAD tools support the interchange of design information using an EDIF file format. The use of EDIF allows for the fault simulation/fault grading tool to be easily integrated into any commercial CAD tool. The use of VHDL as a DUT input file format allows the designer to use the native file format utilized by fault simulation/fault grading tool.

4.3. Process Structure

The low-level requirements of the fault simulation/fault grading tool are presented in Section 2 and Section 3. The mapping of the low-level requirements to a fault simulation/fault grading process is described in this subsection. Specifically, a graphical representation of the process flow associated with the fault simulation/fault grading tool is included as Figure 4.2. The DUT is provided to the fault simulation/fault grading tool as either an EDIF or VHDL file. The DUT model is read by the tool and the fault insertion mechanism is added to the model. The upper leftmost process block in Figure 4.2 generates the VHDL model with fault insertion file. The next process block in Figure 4.2 constructs the complete fault list by reading the VHDL model with fault insertion and applying a user defined fault model. A preprocessing fault equivalence stage is encountered next in Figure 4.2. The preprocessing fault equivalency step determines classes of equivalent faults and removes all but one equivalent fault from each fault class. The reduced set of faults comprises one equivalent fault from each equivalent fault class and is written to a Reduced Fault List (RFL) file. The RFL file is then processed to generate the Working Fault List (WFL) file. The designer must decide whether the fault grading is to be performed using either exhaustive evaluation of all faults in the RFL or a statistical evaluation of faults sampled from the RFL. Both branches of the decision process are depicted as the output of the leftmost decision block in Figure 4.2. If the statistical approach is selected then the RFL is randomly sampled to generate the WFL. If the exhaustive approach is selected the RFL file is copied to generate the WFL file. The loop where the faults contained in the WFL are evaluated is encountered next. The evaluation loop begins by selecting an unevaluated fault from the WFL. The selected fault is then evaluated via fault simulation which utilizes WAVES input/output data. A fault equivalence analysis is performed to determine equivalent detected/undetected faults. The result of the fault simulation and fault equivalence is recorded in a results log file. Additionally, the set of equivalent faults is removed from the WFL. If the WFL contains unevaluated faults then the fault evaluation loop is repeated. If the WFL is empty then the fault grading process is performed. The fault coverage estimate and the set of undetected faults is recorded to file. At this point the fault simulation/fault grading process is complete.

4.4. Applicable fault simulation techniques

The analysis of the attributes associated with existing fault simulation techniques described in [2] is performed to determine which fault simulation methods can be employed for VHDL-based fault simulation. The attributes of fault simulation methods are divided into several categories. The first attribute category is whether the fault simulation technique requires a custom simulation engine or can utilize an existing CAD simulator. VHDL-based fault simulation requires the use of an existing CAD simulator; that is, any VHDL compliant simulator. The second attribute category

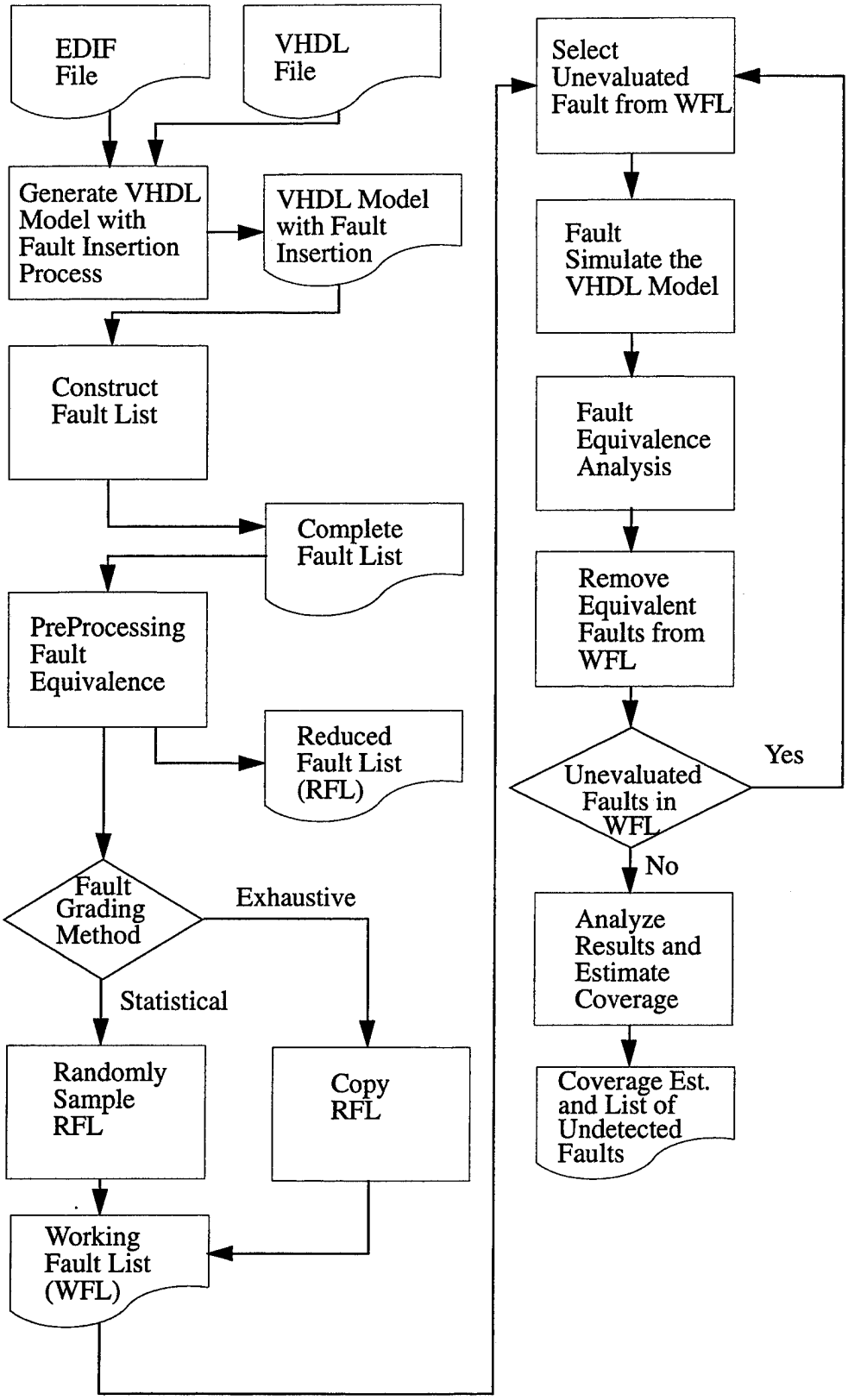


Figure 4.2. Process diagram for the fault grading tool showing all relevant steps.

concerns the ability of the fault simulation technique to support multiple levels of design abstraction. Specifically, the fault simulation technique must be able to insert faults at all levels of design abstraction to satisfy this attribute category. The next two attribute categories deal with whether the fault simulation technique supports combinational and sequential circuits, respectively. The last two attribute categories deal with whether the fault simulation technique requires dynamic memory allocation and custom data structures, respectively. For example, concurrent fault simulation requires a custom data structure to represent each component in the DUT and dynamic memory allocation to store the fault list associated with each component.

An overview of whether a fault simulation technique satisfies each attribute category is included as Table 4.1. The columns of Table 4.1 represent each attribute category while the rows are associated with a specific fault simulation technique. A ✓ symbol in a table cell denotes that a given fault simulation method has a given attribute. For example, the first row of Table 4.1 represents serial fault simulation which has the attribute of using an existing circuit simulator denoted by a ✓ symbol in the second column. Additionally, Table 4.1 represents a summary of requirements for the manner in which the existing set of fault simulators described in literature have been implemented.

Parallel fault simulation is typically used to evaluate gate-level models. Models with higher levels of abstraction such as algorithmic-level models or behavioral microprocessor models are typically not evaluated using parallel fault simulation. High-level models do not map efficiently to

TABLE 4.1. Features and requirements of existing fault simulation techniques

Features and Req./ Simulation Technique	Custom Simulator	Existing Circuit Simulator	Multiple Levels of Design Abstraction	Comb. Circuits	Seq. Circuits	Dynamic Memory Allocation	Data Structures
Serial		✓	✓	✓	✓		
Parallel		✓		✓	✓		
Deductive	✓			✓	✓	✓	✓
Concurrent	✓		✓	✓	✓	✓	✓
Differential		✓			✓		
Hierarchical Concurrent	✓		✓	✓	✓	✓	✓
Hierarchical Serial		✓	✓	✓	✓		

the parallel fault simulation paradigm. Specifically, for parallel fault simulation to be practical the W parallel DUTs must be simulated using W-bit computer words where each bit represents a different DUT. The parallelization process for gate-level models is relatively straight forward. The DUT is transformed into a model comprised of one input NOT gates and two input AND, OR, NOR, NAND and XOR gates. The modified model is then translated to machine instructions that execute on a host processor. Each machine instruction has W-bit input and output operands. The level of effort required to implement a VHDL-based fault simulation tool using a parallel fault simulation technique for gate-level models is small. However, the level of effort required to implement VHDL-based parallel fault simulation for models with higher levels of abstraction is quite high. The primary difficulty is determining an efficient mechanism for translating behavioral VHDL models such as microprocessor model into W parallel behavioral models. Specifically, significant amounts of research and development would be required to implement VHDL-based parallel fault simulation which can be used at all levels of design abstraction. For this reason parallel fault simulation is not a preferred technique for VHDL-based fault simulation.

Deductive and concurrent fault simulation techniques suffer a similar problem. Both concurrent and deductive fault simulation methods propagate fault lists when the internal signals in the DUT are updated. The size of the fault list associated with each signal varies based on the input applied to the DUT and the location of the signal in the DUT. For this reason the fault list associated with each signal is typically stored in a data structure where the memory is dynamically allocated during fault simulation. Specifically, the dynamic memory allocation/deallocation occurs when a signal value is updated during the fault simulation of the DUT. The individual faults in the fault list are typically stored in data structures. The fault list propagation attributes associated with the concurrent and deductive simulation techniques require special simulator features; that is, all existing concurrent and deductive simulators are implemented using a custom simulator. A large amount of research and development effort would be required to determine a VHDL implementation of either the concurrent or deductive fault simulation techniques. For this reason both concurrent and deductive techniques are not a preferred method for VHDL-based fault simulation.

Differential fault simulation also has some fundamental limitations. Specifically, differential fault simulation is primarily used for sequential circuits modeled at the gate-level. The simulation technique is based on evaluating the combinational portion of the DUT for each undetected fault. Each simulation requires that the state of the sequential circuit be restored before the next fault is evaluated. A significant amount of research and development is required for VHDL-based differential fault simulation to determine: (1) a technique to perform state restoration, and (2) a method to extend the differential technique for use with high-level behavioral models. The difficulty associated with solving the aforementioned two issues makes differential fault simulation an undesirable method for VHDL-based fault simulation.

Existing hierarchical fault simulators utilize one of two different fault simulation techniques to inject faults: (1) concurrent fault simulation, and (2) serial fault simulation. Concurrent based hierarchical fault simulation has the same problems associated with concurrent fault simulation. Specifically, the fault list propagation requires the dynamic allocation/deallocation of memory for the fault list data structures. Hierarchical concurrent fault simulation is difficult to implement as a VHDL-based technique because of the development effort associated with the fault list propagation attribute. For this reason, hierarchical concurrent fault simulation is not a preferred technique for VHDL-based fault simulation. Thus, serial based hierarchical fault simulation is the only existing fault simulation technique which requires a reasonable amount of effort to implement in a VHDL-based fault simulator while satisfying all of the requirements specified in Section 2. A more in depth discussion of the background theory associated with the aforementioned fault simulation techniques is presented in [2]. Specifically, the algorithms which define the operation of the fault simulation techniques listed in Table 4.1 are presented in [2].

5. Summary

This document describes the requirements associated with a VHDL-based fault simulation/ grading tool. A summary of the fault simulation requirements described in Section 2, the fault grading requirements presented in Section 3, and the tool structure requirements defined in Section 4 is included in this section as Table 5.1. The first two columns of Table 5.1 describe the requirement while the third column indicates the section number of this document where the requirement is defined. It is envisioned that Table 5.1 will be used to verify that the developed fault simulation/ grading tool meets all design requirements.

TABLE 5.1. Summary of Fault Simulation/Fault Grading Tool Requirements.

Requirement Description	Requirement Attributes	Section No.
Language required to model the DUT	VHDL 1076-1993	Section 2.1
	MVL9 1164-1993	
Fault insertion technique requirement	Consistent fault insertion technique across all levels of modeling abstraction	Section 2.2
	Insertion technique is decoupled from the DUT model	
	Standard VHDL features are used to perform fault insertion	
	Fault insertion technique can be used with any VHDL compliant simulator	
Automated fault simulation	Fault grading requires a large number of fault simulations be performed in an automated fashion	Section 2.3
Interactive fault simulation	Designer must be able to observe the effects of inserted faults	Section 2.4
Fault models required	Support user defined fault models at each level of abstraction	Section 2.5
	Support the stuck-at fault model for gate-level models	
Hierarchical fault simulation	Exploit existing design hierarchy to speed fault simulation	Section 2.6
Test vector language	Test inputs and the output associated with each input vector for a given DUT stored using WAVES 1029.1-1991	Section 2.7
Fault simulation acceleration requirement	Exploit fault equivalency to accelerate the fault grading process	Section 2.8
Fault simulation output requirement	Set of all faults detected during fault simulation	Section 2.9
	Set of all faults undetected by fault simulation	
	Number of detected/undetected faults for fault simulation	
Synthesis requirement	Fault simulate before synthesis	Section 2.10
	Fault simulate VITAL model after synthesis without timing	
	Fault simulate VITAL model after synthesis with timing	
Tool host requirements	Execute on a Sun SPARC workstation	Section 2.11
Hybrid fault simulation	Evaluate hybrid models via fault simulation	Section 2.12
Fault grading requirement	Automated fault grading	Section 3.1
	Fault grading input data format	Section 3.2
	Fault grading performed via MIL-STD 883	Section 3.3
Commercial CAD Tool Independence	Fault simulation/fault grading tool must be readily integrated in existing CAD tools	Section 4.1
Tool input file format	EDIF 3.0.0	Section 4.2
	VHDL 1076-1993	

Bibliography

[1] IEEE Standard VHDL Language Reference Manual, ANSI/IEEE Std. 1076-1993

[2] Smith, D. Todd, Todd A. DeLong, and Barry W. Johnson, "A Survey of Fault Simulation, Fault Grading, and Test Pattern Generation Techniques with an Emphasis on the Feasibility of VHDL Based Fault Simulation", Draft Technical Report, Center for Semicustom Integrated Systems, Department of Electrical Engineering, University of Virginia, January 1996.

[3] IEEE Standard Multivalue Logic System for VHDL Model Interoperability, IEEE Std. 1164-1993

[4] IEEE Standard for Waveform and Vector Exchange (WAVES), IEEE Std. 1029.1-1991

[5] MIL-STD-883D, "Test Methods and Procedures for Microelectronics," Method 5012.1, November 1991

[6] Presentation by Mark Richards, ARPA Technology Kickoff Meeting, RASSP Program, Oct. 21-22, 1993, Arlington VA

APPENDIX

MONTHLY STATUS REPORT

Period: December 1995

Contract No: F30602-95-C-0220
Title: VHDL Fault Simulation and Automatic Test Pattern Generation
Contractor: University of Virginia

1. TASKS PERFORMED

TASK 1: Requirements document for VHDL based fault simulation/fault grading and state of the art fault simulation, fault grading, and ATPG survey.

Status: Work began 1 Oct 95 and is on schedule to be completed on 15 Jan 96.

TASK 2: Fault simulation tool development -- implementation plan, design review, software development

Status: Work is scheduled to begin 1 Jan 96.

TASK 3: Software installation manual

Status: Work is scheduled to begin 1 Jan 97.

TASK 4: Software users manual

Status: Work is scheduled to begin 1 Jan 97.

TASK 5: Evaluation of examples

Status: Work is scheduled to begin 1 Jan 97.

TASK 6: Final technical report

Status: Work is scheduled to begin 1 Oct 97.

2. SIGNIFICANT ACCOMPLISHMENTS

OVERALL: Successfully accomplished the contract kickoff meeting which clearly defined the overall goals and scope of this project.

TASK 1: Completed the requirements document and forwarded final version to Rome Laboratory. The final version of the state of the art survey is scheduled to be completed on 15 January 1996.

3. PROBLEMS ENCOUNTERED

No significant problems have been encountered which impact the project budget and/or schedule.

4. SCHEDULE RECONCILIATION

None.

5. NEXT PERIOD ACTIVITY

TASK 1: Complete the state of the art survey.

TASK 2: Begin writing the fault simulation/fault grading simulation implementation document.

6. BUDGET SUMMARY

Expenditures to date: $11,223.74

7. CONFERENCES AND TRIPS

None.

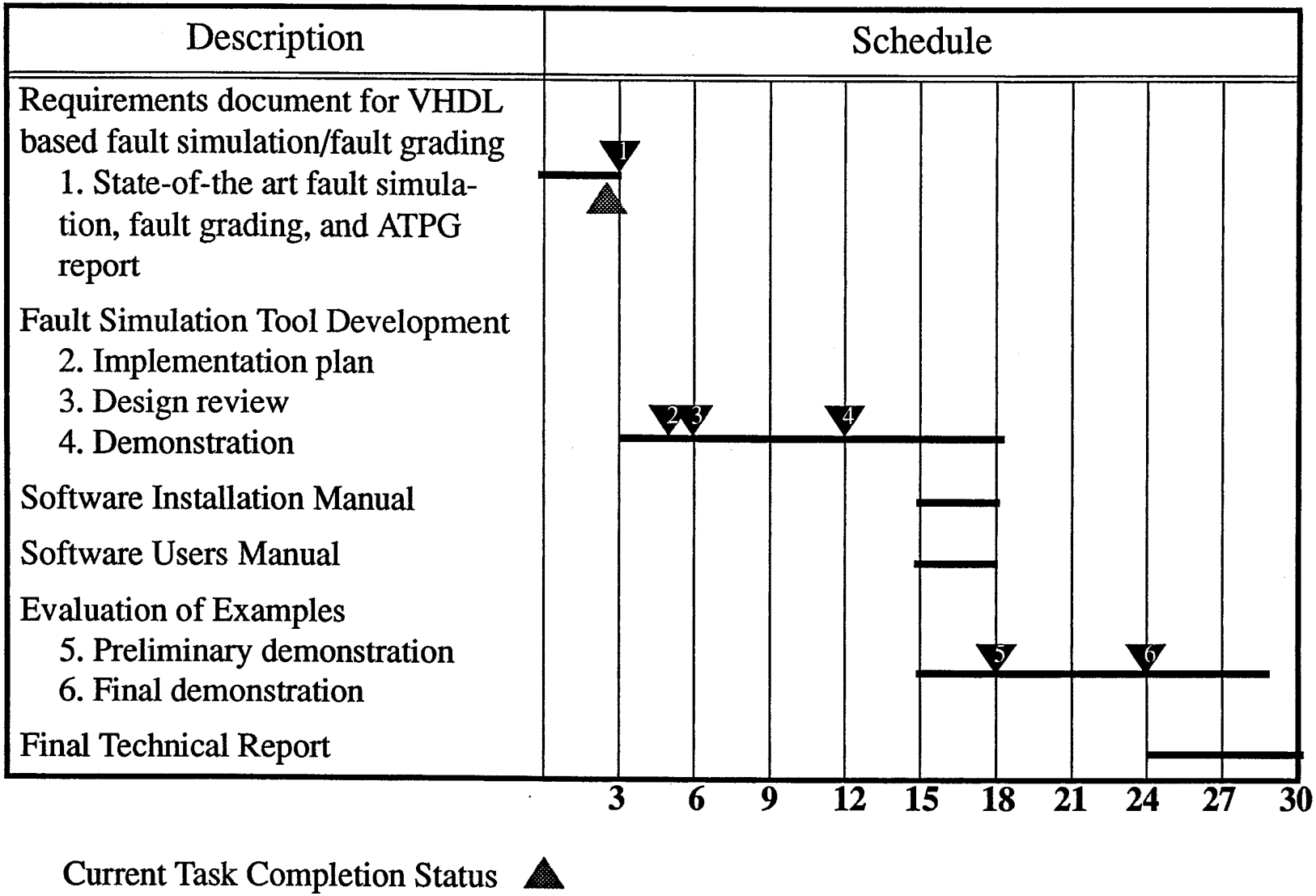
8. OTHER COMMENTS

None.

9. TECHNICAL PAPERS

None.

Current Schedule Status



Current Task Completion Status ▲

MONTHLY STATUS REPORT

Period: November 1995

Contract No: F30602-95-C-0220
Title: VHDL Fault Simulation and Automatic Test Pattern Generation
Contractor: University of Virginia

1. TASKS PERFORMED

TASK 1: Requirements document for VHDL based fault simulation/fault grading and state of the art fault simulation, fault grading, and ATPG survey.

Status: Work began 1 Oct 95 and is on schedule to be completed on 1 Jan 96.

TASK 2: Fault simulation tool development -- implementation plan, design review, software development

Status: Work is scheduled to begin 1 Jan 96.

TASK 3: Software installation manual

Status: Work is scheduled to begin 1 Jan 97.

TASK 4: Software users manual

Status: Work is scheduled to begin 1 Jan 97.

TASK 5: Evaluation of examples

Status: Work is scheduled to begin 1 Jan 97.

TASK 6: Final technical report

Status: Work is scheduled to begin 1 Oct 97.

2. SIGNIFICANT ACCOMPLISHMENTS

OVERALL: Successfully accomplished the contract kickoff meeting which clearly defined the overall goals and scope of this project.

TASK 1: Completed the literature search on the state of the art survey. The literature search revealed over 200 pertinent references. A first draft of the state of the art survey is complete.

3. PROBLEMS ENCOUNTERED

No significant problems have been encountered which impact the project budget and/or schedule.

4. SCHEDULE RECONCILIATION

None.

5. NEXT PERIOD ACTIVITY

TASK 1: Complete the state of the art survey and the fault simulation/fault grading tool requirements document.

6. BUDGET SUMMARY

Expenditures to date: $8,510.10

7. CONFERENCES AND TRIPS

None.

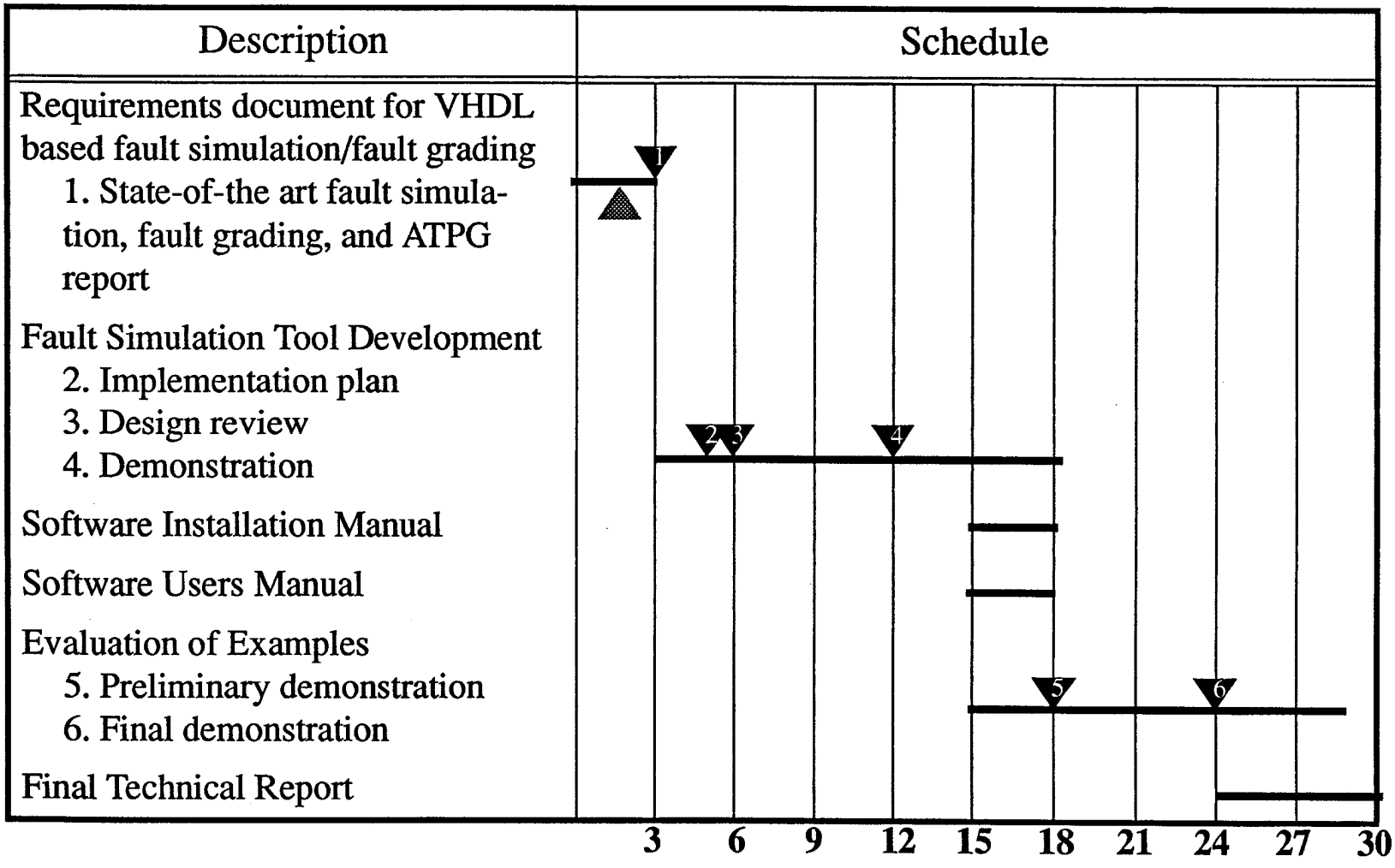
8. OTHER COMMENTS

None.

9. TECHNICAL PAPERS

None.

Current Schedule Status



Current Task Completion Status ▲

DISTRIBUTION LIST

1 - 3	Rome Laboratory/ERM 525 Brooks Rd. Griffiss AFB, New York 13441-4505 Attention: Dr. James P. Hanna
4 - 5	B. W. Johnson
6 - 7	M. Rodeffer, Clark Hall
*	SEAS Postaward Research Administration
8	Defense Technical Information Center Bldg 5., Cameron Station Alexandria, VA 22304-6145
*	Mr. Michael Karp, Administrative Contracting Officer Office of Naval Research Resident Representative 101 Marietta Tower, Suite 2805 101 Marietta Street Atlanta, GA 30303
9	SEAS Preaward Research Administration

*Cover Letter

JO#6832:ph